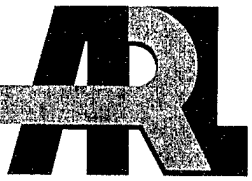
ARMY RESEARCH LABORATORY



SEDRIS Transmittal Generation for a Dismounted Infantry Simulator

by Andrew M. Neiderer

ARL-TR-2801 August 2002

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2801 **August 2002**

SEDRIS Transmittal Generation for a Dismounted Infantry Simulator

Andrew M. Neiderer
Computational and Information Sciences Directorate, ARL

Abstract

Interoperability of heterogeneous computer-based simulators is integral to today's military training systems. This report presents a standardized solution for the distribution of environmental data resulting from a munition perforating an urban structure as computed by the U.S. Army Research Laboratory Dismounted Infantry Simulator (DISim). Synthetic Environment Data Representation and Interchange Specification (SEDRIS) transmittals are generated for such an interaction. Each SEDRIS transmittal contains resulting geometric and attribute data. The application programming interface (API) to write, which is necessary for translating the native data to SEDRIS transmittal format (STF) data, is given. The SEDRIS API language bindings are presently available in the C++ programming language. The developed C++ class STF_Fracture.H encapsulates both data and functions, which are defined in STF_Fracture.C, and allows an instanced variable access to the SEDRIS library for the generation of transmittals.

Acknowledgments

The author would like to thank Mark A. Thomas, also a member of the Distributed Simulation Team. His idea of generation and distribution of Synthetic Environment Data Representation and Interchange Specification (SEDRIS) transmittals by the U.S. Army Research Laboratory Dismounted Infantry Simulator (DISim) motivated this work. He is the author of DISim, which continues to evolve as new technology is added.

INTENTIONALLY LEFT BLANK.

Contents

1. Introduction

The U.S. Army Research Laboratory (ARL) Dismounted Infantry Simulator (DISim) provides for mission rehearsal of military operations on urban terrain (MOUT). Many components for realistic simulations have been integrated, including (1) dynamic terrain, (2) shadows, (3) battlefield smoke and dust, and (4) noise from blasts [1]. A recent addition has been perforation of urban structures [2] and subsequent distribution of resulting rubble [3].

A breaching operation of an urban structure (e.g., a strategically located building) could improve MOUT by providing an alternative entrance to a building interior. This may be necessary since predictable entry, such as through a window or door, might not be an option. Selecting and creating an entry point then results in debris which must be negotiated by mobility units in the urban environment.

The purpose of this report is to describe an approach for DISim distribution of environmental data to Synthetic Environment Data Representation and Interchange Specification (SEDRIS) compliant simulators on a network of computers. The next section provides a brief introduction to SEDRIS technology; further reading materials are recommended in references [1-5]. Section 3 presents our particular application and includes the actual C++ class. Finally, we conclude and note future efforts.

2. SEDRIS Overview

Of the many modeling and simulation (M&S) systems developed within the Department of Defense (DOD), most of these simulators are dependent on a database for representation of the physical world. But interoperability on a computer network is complicated by the fact that the environmental data was never standardized. Thus, a database interchange mechanism was needed.

SEDRIS is a standard for the transmission of environmental data among disparate simulators on a distributed network. The Defense Modeling and Simulation Office (DMSO) is the sponsoring organization for this effort. There are many supporters, or SEDRIS associates, including industry participation, educational institutions, and government agencies. The SEDRIS homepage [6] lists all the registered members.

Four components assist in SEDRIS transmittal production and consumption: (1) the Data Representation Model (DRM), (2) the Spatial Reference Model (SRM), (3) Environmental Data Coding Specification (EDCS), and (4) the Application Program Interface (API) Specfication. Some or all of these are necessary for successful generation of SEDRIS transmittal format (STF) of data. Ultimately, each participant is responsible for providing translation software, to read and/or write, that conforms to the specification.

The first step should be to analyze the native data to identify what needs to be mapped using the SEDRIS DRM. This includes primitives representing the environmental data, their attributes (if any), and the topological relationships that exist among them. During this step, the syntax and structural semantics of the data are developed. The DRM identifies not only the primitive data, but helps identify API classes and the relevant fields. The development of a precise mapping document is critical to a thorough description of the environmental data.

A detailed mapping document is prepared by adding in the SRM and EDCS for a more complete expression of the primitive data. The SRM provides the means to define a unified approach to representing spatial location information. SRM accomplishes this by describing locations and directions through use of a coordinate system. The EDCS aids in identifying environmental objects through characterization and classification of the data. For example, a tree could be represented by the data modeling constructs "Model," "Geometry," and "Feature," which are defined by the API. The result is a structured collection, or library, of data elements organized for SEDRIS descriptive identification within the API.

The third and final step of transmittal generation is the writing of the translation software. Here the API is created and includes opening the transmittal for writing, adding in the primitive data, freeing memory when finished with an object, and finally closing the transmittal. The API defines some 364 classes to assist in STF creation.

Once a transmittal is linked to the SEDRIS core libraries, it should then be validated. Review of member data and its structure can be accomplished using one of several transmittal browsers available from the SEDRIS organization [6] or one of its associate members. We chose "Side-by-Side" by Acusoft, Inc., because of its additional three-dimensional viewer capability [7].

The intent of this section was to simply provide a motivation and framework for our particular application. It is left to the reader to visit the SEDRIS website for a detailed description given by online manuals and the actual API. The remainder of this report discusses STF creation for DISim.

3. STF Generation for DISim

We now present the actual write API that DISim defines for the production of STFs in a simulation environment. This preliminary version concentrates only on DRM and API development; the other two components, namely SRM and EDCS, may be added for a more complete description in future releases. All of the following methods are defined in STF_Fracture.C (see Appendix A).

The native code begins SEDRIS transmittal creation by instantiating a STF_Fracture object. This automatically invokes the default constructor function, which initializes the protected member data. The function create() opens the transmittal for writing. If successful, it proceeds to create top-level objects in the createDescription() method - tokens for Description, Point of Contact, Access, Data Quality, Keywords, Absolute Time Point, Citation, Process, and Transmittal Summary. Appendix B illustrates a tree representation

of an STF_Fracture transmittal. The containers for the image library (IL) and model library (ML) are then determined before adding attributes and primitives.

The texture for the impacted structure is added as a SEDRIS object to the IL. An image mapping function (IMF) for each corresponding primitive will then point to an image. Many flags must be defined when creating an IMF token, or any other token for that matter. Comments in core header files (sedris.h, in particular) assisted in our selection. Note that images can be created all at once or added to the IL as needed. We chose the latter since both attribute and primitive initialization could then be isolated to just one method - addPolygon().

The actual data elements are now added to ML for the environment. An analysis of the native data identified Performer* triangles as the primitive data elements. Each triangle is uniquely identified by three vertices, with a two-dimensional texture coordinate assigned at each vertex. The corresponding IMF is now attached to complete the description of the primitive. This process is repeated for each triangle as needed.

Once the transmittal has been built, the default destructor is called. The destructor is automatically invoked when the class object goes out of scope. This method returns memory to the operating system and closes the SEDRIS transmittal.

Appendix C illustrates a Side-by-Side view of an urban structure perforated by a projectile. Three transmittals resulted - one for the exterior wall of the building, a transmittal representing an interior impact, and one resulting as the resultant projectile exited. Note that the texture is undefined for this third polygonal object.

4. Conclusion

This report defines an API for translating native data in ARL's DISim to STF. STFs are available for consumption by SEDRIS-compliant simulators on a computer network. Details of the API to write are provided in the appendices and should minimize the effort required for retrieval of such data.

Currently, only perforated urban structures are mapped to STFs. An effort is now underway for generating a more comprehensive SEDRIS description of the entire environment.

*Performer is a registered trademark of Silicon Graphics, Inc. (SGI).

INTENTIONALLY LEFT BLANK.

5. References

1. Thomas, M. A. "The ARL Army Experiment 3 Individual Combatant/Military Operations on Urbanized Terrain (MOUT) Demonstration." ARL-TN-102, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, January 1998.
2. Neiderer, A. M., M. A. Thomas, and R. Pearson. "A Fracturing of Polygonal Objects." ARL-TR-1649, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, April 1998.
3. Neiderer, A. M., M. A. Thomas, and C. E. Hansen. "Distribution of Fragments Resulting From Polygonal Object Fracture." ARL-TN-182, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, September 2001.
4. Carswell, J., J. Campos, and R. Cox. *The SEDRIS Transmittal Format.* Science Applications International Corporation, Orlando, FL, 1999.
5. U.S. Army Simulation, Training and Instrumentation Command. *SEDRIS and The Synthetic Environment Domain.* Vol. 1 of the SEDRIS Documentation Set, Orlando, FL, March 1998.
6. SEDRIS homepage. <www.sedris.org>.
7. Acusoft, Inc. <www.acusoft.com/products/sbs>.

INTENTIONALLY LEFT BLANK.

Appendix A. STF_Fracture Class

The C++ class STF_Fracture encapsulates both data and functions for a Synthetic Environment Data Representation and Interchange Specification transmittal of a fractured structure in an urban environment. The method interfaces are declared in the header file STF_Fracture.H and subsequently defined in the source file STF_Fracture.C. Note that a default constructor and destructor are also defined.

```
/* ============================================================================================== */
/*                                                                                                */
/*   File name:  STF_Fracture.H                                                                   */
/*                                                                                                */
/*   Purpose:    class to encapsulate (1) functions and (2) data for                              */
/*                SEDRIS transmittal creation of polygons.                                        */
/*                                                                                                */
/*   By:         Andrew M. Neiderer  26 March, '02.                                               */
/* ============================================================================================== */

#include <stdio.h>

#include <Performer/pf.h>                          // for pfVec3

#include "se_read0.h"                              // for opening and retrieving data from SEDRIS transmittals
#include "se_write0.h"                             // for creating and modifying SEDRIS transmittals

class STF_Fracture {
  public:                                          // interface
                                                   //   (1) manager functions
    STF_Fracture()                                 //     (a) default constructor function
    {
#ifdef Debug
fprintf(stdout, "    STF_Fracture()\n\n");
#endif

      _impl_id = "stf";
      _xmittal = NULL;
      _xmittal_name = NULL;
      _root = _model_lib = _union = _image_lib = _image = NULL;
      _model_id = 1;
      _image_id = 1;
    }

    ~STF_Fracture()                                //     (b) default destructor function
    {
#ifdef Debug
fprintf(stdout, "    ~STF_Fracture()\n\n");
#endif

      SE_FreeObject(_union);
      SE_FreeObject(_model_lib);
      SE_FreeObject(_image);
      SE_FreeObject(_image_lib);
      SE_FreeObject(_root);

      if ( SE_CloseTransmittal(_xmittal) != SE_SUCCESS )
        printf("Error: failed to close transmittal\n");
    }

                                                   //   (2) implementor functions
```

```
      int createDescription();

      int createImageLib();

      int addImage(unsigned int *, int, int, int);

      int createModelLib();

      int addPolygons(pfVec3 *,
                      pfVec2 *, pfTexture *,
                      int);

      int create(char *,
                 pfVec3 *,
                 pfVec2 *, pfTexture *,
                 int);

    private:

      int createObj(SE_TOKEN_ENUM,
                    SE_OBJECT *,
                    SE_FIELDS * = NULL);
      int addObj(SE_OBJECT,
                 SE_OBJECT,
                 SE_FIELDS * = NULL,
                 SE_OBJECT = NULL,
                 SE_FIELDS * = NULL);
      int freeObj(SE_OBJECT);

    protected:
      char           *_impl_id;
      SE_TRANSMITTAL _xmittal;
      char           *_xmittal_name;
      SE_OBJECT      _root,
                     _model_lib,
                     _union,
                     _image_lib,
                     _image;
      int            _model_id;
      int            _image_id;
  };
```

```
//      (a) create top level objects
//
//      (b) create Image Library
//
//      (c) add an Image to Image Library
//
//      (d) create Model Library and add a Model
//
//      (e) add Polygons to Model's union of geometry
//
//
//
//      (f) open SEDRIS transmittal, then
//           create description, Image Library, and Model Library.
//
//
//
//
//    (3) helping functions
//      (a) create an object
//
//
//      (b) add an object
//
//
//
//
//
//      (c) release an object
//
// representation
//   implementation identifier
//   transmittal ptr
//   transmittal name
//   Transmittal Root
//   Model Library
//   union of Primitive Geometry
//   Image Library
//   Image for texture
//   model identifier
//   image identifier
```

```
/* ========================================================================== */
/*                                                                            */
/*   File name: STF_Fracture.C                                                */
/*                                                                            */
/*   Purpose:   implementation of member functions, ie methods, of STF_Fracture class    */
/*                                                                            */
/*   Functions, parameters:                                                   */
/*               (1) createObj - create an object                             */
/*               (2) addObj - add an object                                   */
/*               (3) freeObj - release an object                              */
/*               (4) createDescription - create top level objects             */
/*               (5) createImageLib - create Image Library                    */
/*               (6) addImage - add an Image to Image Library                 */
/*               (7) createModelLib - create Model Library and add a Model    */
/*               (8) addPolygons - add polygons to Model's union of geometry  */
/*               (9) create - open SEDRIS transmittal, then create description, Image    */
/*                            Library, and Model Library.                     */
/*                                                                            */
/*   By:        Andrew M. Neiderer 26 March '02                               */
/* ========================================================================== */

#include <stdio.h>
#include <iostream.h>
#include <string.h>                     // for strlen()
#include <malloc.h>                     // for malloc()

#include <Performer/pf.h>               // for pfVec3
#include <Performer/pr/pfTexture.h>     // for pfTexture::getImage()

#include "STF_Fracture.H"               // for STF_Fracture class definition

/* ************************************************************ */
/* createObj (type, obj, obj_f)                                 */
/*   SE_TOKEN_ENUM type      (in) -                             */
/*   SE_OBJECT *obj          (in) - object                      */
/*   SE_FIELDS *obj_f        (in) - object fields               */
/*                                                              */
/* Create an object.                                            */
/*                                                              */
/* Return true (non-zero) or false (zero).                      */
/* ************************************************************ */

int STF_Fracture::createObj(SE_TOKEN_ENUM type,
                            SE_OBJECT *obj,
                            SE_FIELDS *obj_f)
{

#ifdef Debug
fprintf(stdout, "    STF_Fracture::createObj()\n\n");
#endif
```

```
    if ( SE_CreateObject(_impl_id,type,obj) != SE_SUCCESS ) {
      printf("Error: Unable to create object\n");
      return 0;
    }
    else if ( SE_AddToTransmittal(*obj,_xmittal) != SE_SUCCESS ) {
      printf("Error: Unable to add object to transmittal\n");
      return 0;
    }
    else if ( obj_f  &&  SE_SetFieldsToDefault(type,obj_f) != SE_SUCCESS ) {
      printf("Error: Unable to set default fields\n");
      return 0;
    }

    return 1;
  }

  /* ************************************************************** */
  /* addObj (aggr, comp, comp_f, link, link_f)                      */
  /*   SE_OBJECT aggr          (in) - aggregate                     */
  /*   SE_OBJECT comp          (in) - component                     */
  /*   SE_FIELDS *comp_f       (in) - component fields              */
  /*   SE_OBJECT link          (in) - link                          */
  /*   SE_FIELDS *link_f       (in) - link fields                   */
  /*                                                                */
  /* Add an object.                                                 */
  /*                                                                */
  /* Return true (non-zero) or false (zero).                        */
  /* ************************************************************** */

  int STF_Fracture::addObj(SE_OBJECT aggr,
                           SE_OBJECT comp,
                           SE_FIELDS *comp_f,
                           SE_OBJECT link,
                           SE_FIELDS *link_f)
  {

  #ifdef Debug
  fprintf(stdout, "    STF_Fracture::addObj()\n\n");
  #endif

    if ( (comp_f  &&  SE_PutFields(comp,comp_f) != SE_SUCCESS)  ||
         (link  &&  link_f  &&  SE_PutFields(link,link_f) != SE_SUCCESS)  ||
         SE_AddComponentRelationship(aggr,comp,link) != SE_SUCCESS ) {
      printf("Error: couldn't complete addObj()\n");
      return 0;
    }

    return 1;
  }

  /* ************************************************************** */
```

```
/* freeObj (obj)                                                    */
/*   SE_OBJECT obj          (in) - object                           */
/*                                                                  */
/* Release an object.                                               */
/*                                                                  */
/* Return true (non-zero) or false (zero).                          */
/* ************************************************************** */

int STF_Fracture::freeObj(SE_OBJECT obj)
{

#ifdef Debug
fprintf(stdout, "    STF_Fracture::freeObj()\n\n");
#endif

  if ( SE_FreeObject(obj) != SE_SUCCESS ) {
    printf("Error: free object failed\n");
    return 0;
  }

  return 1;
}

/* ************************************************************** */
/* createDescription ()                                             */
/*                                                                  */
/* Create top level objects - Description, Point of Contact, Access,  */
/*  Data Quality, Keywords, Absolute Time Point, Citation, Process,   */
/*  Transmittal Summary.                                            */
/*                                                                  */
/* Return true (non-zero) or false (zero).                          */
/* ************************************************************** */

int STF_Fracture::createDescription()
{
  SE_OBJECT obj;
  SE_FIELDS f;
  char *s;

#ifdef Debug
fprintf(stdout, "    STF_Fracture::createDescription()\n\n");
#endif

  // Transmittal Root

  if ( !createObj(SE_TRANSMITTAL_ROOT_TOKEN,&_root,&f) )
    return 0;

  f.u.Transmittal_Root.name.string_length = strlen(_xmittal_name);
  f.u.Transmittal_Root.name.string_value = _xmittal_name;
```

```
if ( SE_PutFields(_root,&f) != SE_SUCCESS  ||
     SE_SetRootObject(_xmittal,_root,&obj) != SE_SUCCESS ) {
  printf("Error: failed to add Transmittal Root\n");
  return 0;
}

// Description

createObj(SE_DESCRIPTION_TOKEN,&obj,&f);
s = "Evaluation";
f.u.Description.abstract.string_value = s;
f.u.Description.abstract.string_length = strlen(s);
addObj(_root,obj,&f);
freeObj(obj);

// Point of Contact

createObj(SE_POINT_OF_CONTACT_TOKEN,&obj,&f);
s = "Army Research Laboratory";
f.u.Point_of_Contact.organization.string_value = s;
f.u.Point_of_Contact.organization.string_length = strlen(s);
s = "SEDRIS customer";
f.u.Point_of_Contact.person_or_position.string_value = s;
f.u.Point_of_Contact.person_or_position.string_length = strlen(s);
s = "APG, MD";
f.u.Point_of_Contact.address.string_value = s;
f.u.Point_of_Contact.address.string_length = strlen(s);
s = "410-278-3203";
f.u.Point_of_Contact.voice_phone.string_value = s;
f.u.Point_of_Contact.voice_phone.string_length = strlen(s);
addObj(_root,obj,&f);
freeObj(obj);

// Access

createObj(SE_ACCESS_TOKEN,&obj,&f);
addObj(_root,obj,&f);
freeObj(obj);

// Data Quality

createObj(SE_DATA_QUALITY_TOKEN,&obj,&f);
f.u.Data_Quality.fictional = SE_TRUE;
addObj(_root,obj,&f);
freeObj(obj);

// Keywords

createObj(SE_KEYWORDS_TOKEN,&obj,&f);
addObj(_root,obj,&f);
freeObj(obj);
```

```
    // Absolute Time Point

    createObj(SE_ABSOLUTE_TIME_POINT_TOKEN,&obj,&f);
    f.u.Absolute_Time_Point.time_significance = SE_CREATION_DATE;
    f.u.Absolute_Time_Point.year = 2002;
    f.u.Absolute_Time_Point.month = SE_FEBRUARY;
    f.u.Absolute_Time_Point.day = 20;
    f.u.Absolute_Time_Point.hour = 15;
    f.u.Absolute_Time_Point.minutes = 0;
    f.u.Absolute_Time_Point.seconds = 0;
    addObj(_root,obj,&f);
    freeObj(obj);

    // Citation

    createObj(SE_CITATION_TOKEN,&obj,&f);
    s = "Fracture Database";
    f.u.Citation.title.string_value = s;
    f.u.Citation.title.string_length = strlen(s);
    s = "ARL/CISD, APG";
    f.u.Citation.originators.string_value = s;
    f.u.Citation.originators.string_length = strlen(s);
    addObj(_root,obj,&f);
    freeObj(obj);

    // Process

    createObj(SE_PROCESS_TOKEN,&obj,&f);
    s = "Pre-process";
    f.u.Description.abstract.string_value = s;
    f.u.Description.abstract.string_length = strlen(s);
    addObj(_root,obj,&f);
    freeObj(obj);

    // Transmittal Summary

    createObj(SE_TRANSMITTAL_SUMMARY_TOKEN,&obj,&f);
    f.u.Transmittal_Summary.geometry_present = SE_PRESENT_IN_MODELS;
    f.u.Transmittal_Summary.models_present = SE_TRUE;
    addObj(_root,obj,&f);
    freeObj(obj);

    return 1;
}

/* ************************************************************ */
/* createImageLib ()                                            */
/*                                                              */
/* Create Image Library.                                        */
/*                                                              */
```

```
/* Return true (non-zero) or false (zero).                          */
/* ***************************************************************** */

int STF_Fracture::createImageLib()
{
  SE_FIELDS f;

#ifdef Debug
fprintf(stdout, "    STF_Fracture::createImageLib()\n\n");
#endif

  createObj(SE_IMAGE_LIBRARY_TOKEN,&_image_lib,&f);
  addObj(_root,_image_lib,&f);

  return 1;
}

/* ***************************************************************** */
/* addImage (data, xSize, ySize, components)                         */
/*   unsigned int *data              (in) - image data               */
/*   int xSize                       (in) - horizontal size          */
/*   int ySize                       (in) - vertical size            */
/*   int components                  (in) - no. of 8-bit components  */
/*                                          per pixel.               */
/*                                                                   */
/* Add Image to Image Library.                                       */
/*                                                                   */
/* Return true (non-zero) or false (zero).                          */
/* ***************************************************************** */

int STF_Fracture::addImage(unsigned int *data,
                           int xSize, int ySize, int components)
{
  SE_FIELDS f;
  char *s;

#ifdef Debug2
fprintf(stdout, "    STF_Fracture::addImage()\n\n");
#endif

  if ( _image != (SE_OBJECT) NULL )
    freeObj(_image);
  createObj(SE_IMAGE_TOKEN,&_image,&f);

  f.u.Image.ID = _image_id;
  s = "Texture";
  f.u.Image.name.string_value = s;
  f.u.Image.name.string_length = strlen(s);
  f.u.Image.color_model = SE_RGB_MODEL;
  f.u.Image.level_count = 1;
  f.u.Image.mip_fields_array = (SE_IMAGE_MIP_FIELDS *) calloc(1, sizeof(SE_IMAGE_MIP_FIELDS));
```

```
    f.u.Image.mip_fields_array->size_horizontal = xSize;
    f.u.Image.mip_fields_array->size_vertical = ySize;
    f.u.Image.mip_fields_array->size_z = 1;
    f.u.Image.scan_direction = SE_IMAGE_SCAN_RU;
    f.u.Image.scan_direction_z = SE_IMAGE_SCAN_Z_NO_Z;
    f.u.Image.component_data_type = SE_UNSIGNED_INTEGER;
  //  f.u.Image.data_is_little_endian = SE_TRUE;

    switch (components) {
      case 1:
        f.u.Image.data_is_3D = SE_FALSE;
        f.u.Image.image_signature = SE_IMAGE_SIGNATURE_LUMINANCE;
        f.u.Image.bits_of_alpha = 0;
        f.u.Image.bits_of_luminance = 8;
        f.u.Image.bits_of_first_color_coord = 0;
        f.u.Image.bits_of_second_color_coord = 0;
        f.u.Image.bits_of_third_color_coord = 0;
        f.u.Image.max_value_of_alpha = 0;
        f.u.Image.max_value_of_luminance = 255;
        f.u.Image.max_value_of_first_color_coord = 0;
        f.u.Image.max_value_of_second_color_coord = 0;
        f.u.Image.max_value_of_third_color_coord = 0;

        break;
      case 2:
        f.u.Image.data_is_3D = SE_FALSE;
        f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
        f.u.Image.bits_of_alpha = 0;
        f.u.Image.bits_of_luminance = 0;
        f.u.Image.bits_of_first_color_coord = 8;
        f.u.Image.bits_of_second_color_coord = 8;
        f.u.Image.bits_of_third_color_coord = 0;
        f.u.Image.max_value_of_alpha = 0;
        f.u.Image.max_value_of_luminance = 0;
        f.u.Image.max_value_of_first_color_coord = 255;
        f.u.Image.max_value_of_second_color_coord = 255;
        f.u.Image.max_value_of_third_color_coord = 0;

        break;
      case 3:
        f.u.Image.data_is_3D = SE_TRUE;
        f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
        f.u.Image.bits_of_alpha = 0;
        f.u.Image.bits_of_luminance = 0;
        f.u.Image.bits_of_first_color_coord = 8;
        f.u.Image.bits_of_second_color_coord = 8;
        f.u.Image.bits_of_third_color_coord = 8;
        f.u.Image.max_value_of_alpha = 0;
        f.u.Image.max_value_of_luminance = 0;
        f.u.Image.max_value_of_first_color_coord = 255;
        f.u.Image.max_value_of_second_color_coord = 255;
```

```
        f.u.Image.max_value_of_third_color_coord = 255;

        break;
      case 4:
        f.u.Image.data_is_3D = SE_FALSE;
        f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
        f.u.Image.bits_of_alpha = 8;
        f.u.Image.bits_of_luminance = 0;
        f.u.Image.bits_of_first_color_coord = 8;
        f.u.Image.bits_of_second_color_coord = 8;
        f.u.Image.bits_of_third_color_coord = 8;
        f.u.Image.max_value_of_alpha = 8;
        f.u.Image.max_value_of_luminance = 0;
        f.u.Image.max_value_of_first_color_coord = 255;
        f.u.Image.max_value_of_second_color_coord = 255;
        f.u.Image.max_value_of_third_color_coord = 255;

        break;
      default:
        f.u.Image.data_is_3D = SE_TRUE;
        f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
        f.u.Image.bits_of_alpha = 0;
        f.u.Image.bits_of_luminance = 0;
        f.u.Image.bits_of_first_color_coord = 8;
        f.u.Image.bits_of_second_color_coord = 8;
        f.u.Image.bits_of_third_color_coord = 8;
        f.u.Image.max_value_of_alpha = 0;
        f.u.Image.max_value_of_luminance = 0;
        f.u.Image.max_value_of_first_color_coord = 255;
        f.u.Image.max_value_of_second_color_coord = 255;
        f.u.Image.max_value_of_third_color_coord = 255;

        break;
    }
    addObj(_image_lib,_image,&f);

    SE_PutImageData(_image,0,0,0,xSize - 1,ySize - 1,0,0,xSize * ySize * components,(unsigned char *) data);

    return 1;
  }

  /* ************************************************************ */
  /* createModelLib ()                                            */
  /*                                                              */
  /* Create Model Library and add a Model.                        */
  /*                                                              */
  /* Return true (non-zero) or false (zero).                      */
  /* ************************************************************ */

  int STF_Fracture::createModelLib()
  {
```

```
    SE_OBJECT obj, obj2;
    SE_FIELDS f;
    char *s;

  #ifdef Debug
  fprintf(stdout, "    STF_Fracture::createModelLib()\n\n");
  #endif

    createObj(SE_MODEL_LIBRARY_TOKEN,&_model_lib,&f);
    addObj(_root,_model_lib,&f);

    createObj(SE_MODEL_TOKEN,&obj,&f);
    f.u.Model.ID = _model_id;
    s = "Fracture";
    f.u.Model.name.string_value = s;
    f.u.Model.name.string_length = strlen(s);
    f.u.Model.model_reference_type = SE_ROOT_MODEL;
    f.u.Model.srf_params.is_2D = SE_FALSE;
    f.u.Model.srf_params.u.params_3d.spatial_reference_frame = SRM_LSR_3D_SRF;
    f.u.Model.srf_params.u.params_3d.u.lsr_parameters.up_direction =
      SE_DIRECTION_OF_UP_POSITIVE_TERTIARY_AXIS;
    f.u.Model.srf_params.u.params_3d.u.lsr_parameters.forward_direction =
      SE_DIRECTION_OF_FORWARD_POSITIVE_SECONDARY_AXIS;
    addObj(_model_lib,obj,&f);

    createObj(SE_GEOMETRY_MODEL_TOKEN,&obj2,&f);
    addObj(obj,obj2,&f);

    freeObj(obj);

    createObj(SE_UNION_OF_PRIMITIVE_GEOMETRY_TOKEN,&_union,&f);
    addObj(obj2,_union,&f);

    freeObj(obj2);

    return 1;
  }

  /* ************************************************************** */
  /* addPolygons (verts, tcoords, texture, nverts)                  */
  /*   pfVec3 *verts                (in) - vertices                 */
  /*   pfVec2 *tcoords              (in) - texture coords at vertices */
  /*   pfTexture *texture           (in) - Performer texture        */
  /*   int nverts                   (in) - no. of vertices          */
  /*                                                                */
  /* Add polygons to Model's union of geometry.                     */
  /*                                                                */
  /* Return true (non-zero) or false (zero).                        */
  /* ************************************************************** */

  int STF_Fracture::addPolygons(pfVec3 *verts,
```

```
                                pfVec2 *tcoords, pfTexture *texture,
                                int nverts)
{
  // SEDRIS API

  unsigned char *data;                    // image
  SE_OBJECT poly,
            vert,
            loc, tex, clr,
            IMF;
  SE_FIELDS f;

  // Performer API

  unsigned int *image;                    // array of texels
  int components;                         // no. of components per pixel
  int ns,                                 // no. of texels in s dimension of image
      nt,                                 // no. of texels in t dimension of image
      nr;                                 // no. of texels in r dimension of image

  int i, j;                               // counters

#ifdef Debug
fprintf(stdout, "    STF_Fracture::addPolygons()\n\n");
#endif

  // Performer texture

  if ( texture != NULL ) {
    texture->getImage(&image,&components,&ns,&nt,&nr);

    // add to Image Library
    addImage(image,ns,nt,components);

#ifdef Debug
    cerr << "      addPolygons : addImage complete " << texture->getName() << endl;
#endif
  }
  else {
#ifdef Debug
    cerr << "      addPolygons : no texture for this polygon" << endl;
#endif
  }

  // each polygon has 3 vertices, ie triangle

  for ( i = 0; i < nverts; i += 3 ) {
#ifdef Debug
fprintf(stdout, "      polygon %d\n\n",i / 3);
#endif
```

```
      createObj(SE_POLYGON_TOKEN,&poly,&f);
      addObj(_union,poly,&f);

      // Image Mapping Function

      if ( texture != NULL ) {
        createObj(SE_IMAGE_MAPPING_FUNCTION_TOKEN,&IMF,&f);
        f.u.Image_Mapping_Function.image_id = _image_id;
        f.u.Image_Mapping_Function.image_mapping_method = SE_REPLACE_IMAGE;
        f.u.Image_Mapping_Function.image_wrap_s = SE_REPEAT_IMAGE;
        f.u.Image_Mapping_Function.image_wrap_t = SE_REPEAT_IMAGE;
        f.u.Image_Mapping_Function.image_projection_type = SE_PLANAR_IMAGE_PROJECTION;
        f.u.Image_Mapping_Function.intensity_level = 1;
        f.u.Image_Mapping_Function.gain = 0;
        f.u.Image_Mapping_Function.image_detail_mapping = SE_FALSE;
        f.u.Image_Mapping_Function.presentation_domain = SE_OTW;
#ifdef Debug
        fprintf(stdout,"      Image_Mapping_Function.image_id = %d\n\n",_image_id);
#endif
        addObj(poly,IMF,&f);
        SE_AddAssociateRelationship(IMF,_image,NULL,SE_FALSE);
        freeObj(IMF);
      }

      // at each vertex, Cartesian coords and texture coords

      for ( int k = 0; k < 3; k++ ) {
        createObj(SE_VERTEX_TOKEN,&vert,&f);
        addObj(poly,vert,&f);
        createObj(SE_LSR_LOCATION_3D_TOKEN,&loc,&f);
        f.u.LSR_Location_3D.x = verts[i + k].vec[0];
        f.u.LSR_Location_3D.y = verts[i + k].vec[1];
        f.u.LSR_Location_3D.z = verts[i + k].vec[2];
        addObj(vert,loc,&f);

        createObj(SE_RGB_COLOR_TOKEN,&clr,&f);
        f.u.RGB_Color.rgb_data.red = 1.0;
        f.u.RGB_Color.rgb_data.green = 1.0;
        f.u.RGB_Color.rgb_data.blue = 1.0;
        addObj(vert,clr,&f);
        freeObj(clr);

        if ( texture != NULL ){
          createObj(SE_TEXTURE_COORDINATE_TOKEN,&tex,&f);
          f.u.Texture_Coordinate.s = tcoords[i + k].vec[0];
          f.u.Texture_Coordinate.t = tcoords[i + k].vec[1];
          addObj(vert,tex,&f);
          freeObj(tex);

        }
```

```
        freeObj(loc);
        freeObj(vert);
      }

      freeObj(poly);
    }

    if ( texture != NULL )
      _image_id++;

    return 1;
  }

  /* ************************************************************** */
  /* create (fileName, verts, tcoords, texture, nverts)             */
  /*   char *fileName                (in) - SEDRIS transmittal      */
  /*   pfVec3 *verts                 (in) - vertices                */
  /*   pfVec2 *tcoords               (in) - texture coords at vertices */
  /*   pfTexture *texture            (in) - Performer texture       */
  /*   int nverts                    (in) - no. of vertices         */
  /*                                                                */
  /* Open SEDRIS transmittal.  Add description, Image Library, and  */
  /*   Model Library.                                               */
  /*                                                                */
  /* Return true (non-zero) or false (zero).                        */
  /* ************************************************************** */

  int STF_Fracture::create(char *fileName,
                           pfVec3 *verts,
                           pfVec2 *tcoords, pfTexture *texture,
                           int nverts)
  {
    char buff[1000];

  #ifdef Debug
  fprintf(stdout, "    STF_Fracture::create()\n\n");
  #endif

    if ( SE_OpenTransmittalByFile(fileName,_impl_id,SE_CREATE,&_xmittal) != SE_SUCCESS ) {
      printf("Error: can't create \"%s\"\n",fileName);
      return 0;
    }

    sprintf(buff,"urn:x-sedris:%s:1",fileName);

    if ( SE_SetTransmittalName(_xmittal,buff) != SE_SUCCESS ) {
      printf("Error: failed to set transmittal name\n");
      return 0;
    }

    _xmittal_name = fileName;
```

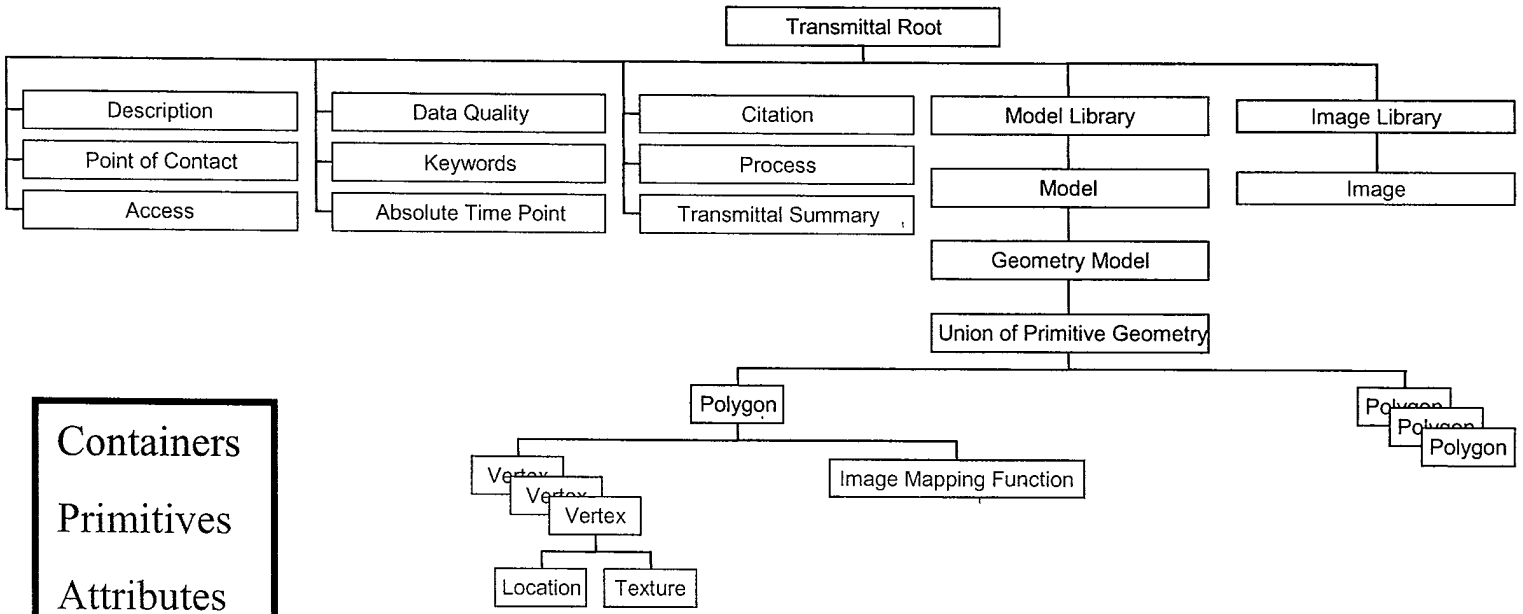
```
    if ( !createDescription()  ||  !createImageLib()  ||
         !createModelLib()  ||  !addPolygons(verts,tcoords,texture,nverts) )
      return 0;

    return 1;
  }
```

Appendix B. Tree Representation of STF_Fracture Transmittal

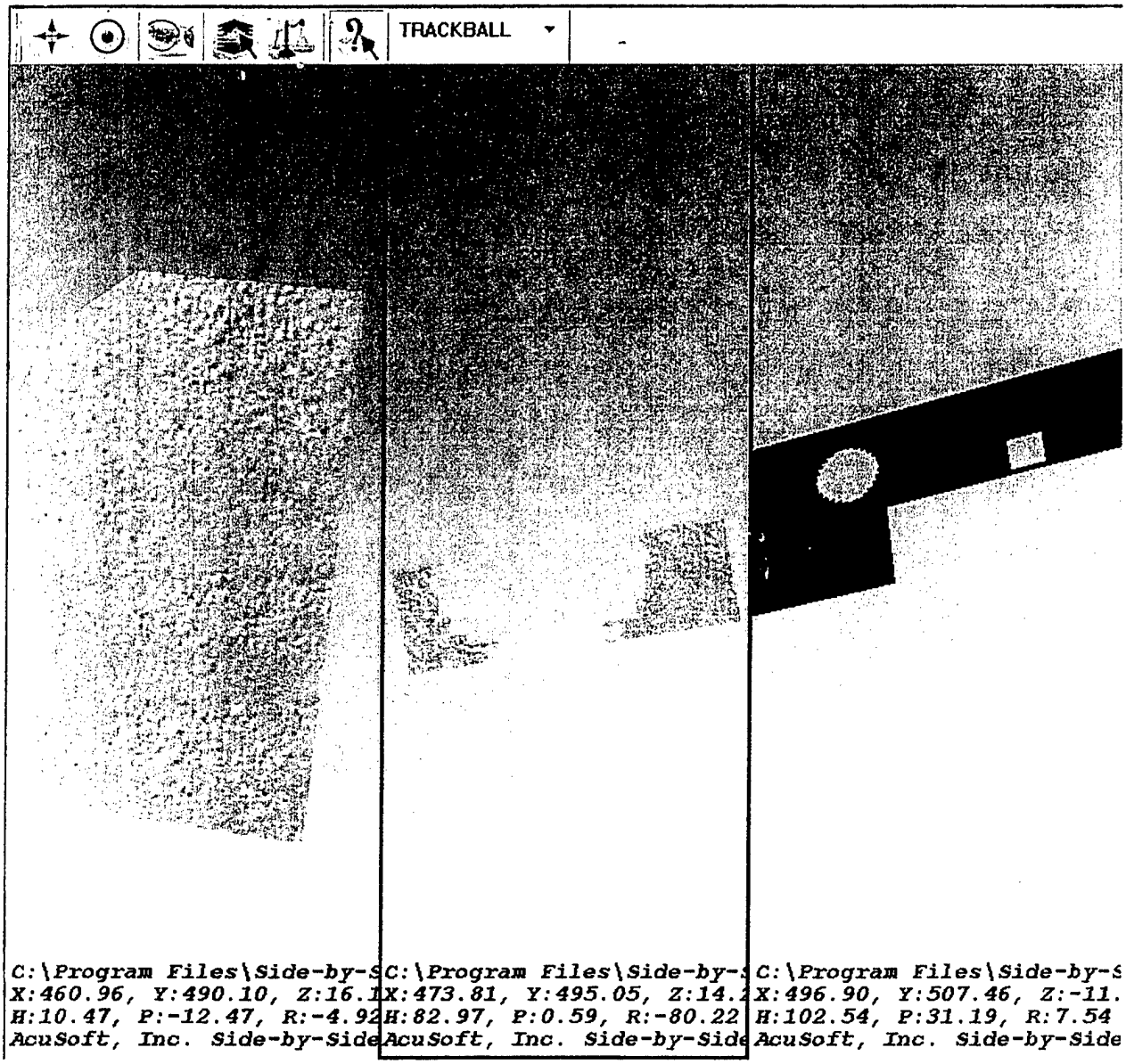
The following diagram organizes a Synthetic Environment Data Representation and Interchange Specification transmittal in hierarchical format. A SEDRIS transmittal format is generated for each fracture and persists during the simulation.

STF_Fracture Transmittal



Appendix C. Example Side-by-Side Display of STF_Fracture Transmittals

The following is a Side-by-Side display of the three transmittals resulting from perforation of an urban structure in Dismounted Infantry Simulator. Note that a texture is not defined in the third illustration.


TRACKBALL
C:\Program Files\Side-by-S
X:460.96, Y:490.10, Z:16.1
H:10.47, P:-12.47, R:-4.92
AcuSoft, Inc. Side-by-Side
C:\Program Files\Side-by-S
X:473.81, Y:495.05, Z:14.2
H:82.97, P:0.59, R:-80.22
AcuSoft, Inc. Side-by-Side
C:\Program Files\Side-by-S
X:496.90, Y:507.46, Z:-11.
H:102.54, P:31.19, R:7.54
AcuSoft, Inc. Side-by-Side

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2002	Final, 1 October 2001–1 March 2002

4. TITLE AND SUBTITLE
SEDRIS Transmittal Generation for a Dismounted Infantry Simulator

5. FUNDING NUMBERS
622120.H16

6. AUTHOR(S)
Andrew M. Neiderer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
AMSRL-CI-CT
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2801

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT*(Maximum 200 words)*

Interoperability of heterogeneous computer-based simulators is integral to today's military training systems. This report presents a standardized solution for the distribution of environmental data resulting from a munition perforating an urban structure as computed by the U.S. Army Research Laboratory Dismounted Infantry Simulator (DISim). Synthetic Environment Data Representation and Interchange Specification (SEDRIS) transmittals are generated for such an interaction. Each SEDRIS transmittal contains resulting geometric and attribute data. The application programming interface (API) to write, which is necessary for translating the native data to SEDRIS transmittal format (STF) data, is given. The SEDRIS API language bindings are presently available in the C++ programming language. The developed C++ class STF_Fracture.H encapsulates both data and functions, which are defined in STF_Fracture.C, and allows an instanced variable access to the SEDRIS library for the generation of transmittals.

14. SUBJECT TERMS	15. NUMBER OF PAGES
SEDRIS transmittal, distributed simulation, data representation model, application programming, fracture	30
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

INTENTIONALLY LEFT BLANK.